UNIVERSITE DE RENNES 1

ENSSAT LSI 1

DS Microprocesseurs ARM

Jeudi 3 février 2005 Durée 1 heure

ENSSAT

Documents de cours, TD, TP autorisés

Le barême est donné à titre indicatif

Il sera tenu compte de la clarté des réponses

---

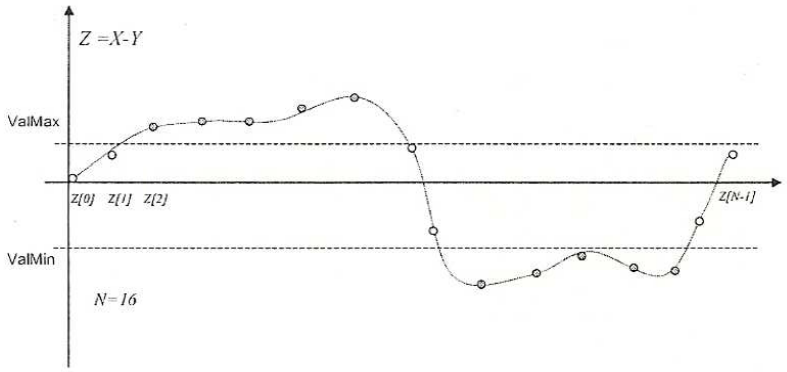
1 Problème I - Développement de code assembleur pour le processeur ARM (···/6,5pt)

Soient X et Y deux vecteurs composés respectivement de N nombres réels codés sur 32 bits. L'objectif est d'analyser la similitude entre ces deux vecteurs. Pour cela nous calculons pour chaque élément i des vecteurs X et Y, la différence entre ces deux vecteurs. Soit le vecteur Z correspondant à la différence entre ces deux vecteurs :

$$Z_i = X_i - Y_i \qquad i \in [0; N-1] \tag{1}$$

Ce vecteur sera sauvegardé en mémoire. Ensuite nous comptons le nombre K de points de ce vecteur Z compris dans l'intervalle [ValMin ; ValMax]. Ces deux valeurs correspondent à des constantes définies dans le code et pouvant être directement utilisées. La routine d'analyse de la similitude devra renvoyer le nombre de points présents dans l'intervalle [ValMin ; ValMax].

Un exemple de vecteur Z contenant 16 éléments est proposé à la figure suivante. Pour cette exemple, le nombre de points présents dans l'intervalle est égal à 6.



Les conventions suivantes sont utilisées pour l'interface avec cette routine :

- Entrée
  - `r0` : adresse du vecteur X
  - `r1` : adresse du vecteur Y
  - `r2` : adresse du vecteur Z
  - `r3` : nombre d'éléments des vecteurs
- Sortie
  - `r0` : nombre de points présents dans l'intervalle.

Nous disposons du code suivant pour tester cette routine (`AnalyseSimilitude`) permettant de déterminer la similitude entre deux vecteurs :

```
#define N      128
#define Nlog2   7

int X[N] = {....}
```

```
int Y[N] = {....}
int Z[N] = {....}


???? AnalyseSimilitude(????);


int main(void)
{
int NbPtsInt;


     AnalyseSimilitude(????);

   while(1);
}
```

Q.1.1) Donner le prototype C de la routine `AnalyseSimilitude` et l'appel en C de cette routine. Vous utiliserez les conventions de passage de paramètres associées au processeur ARM.

Q.1.2) Écrire le code C de la routine `AnalyseSimilitude`.

Q.1.3) Écrire le code assembleur (avec commentaires) de cette routine en utilisant les conventions définies ci-dessus.

2 Problème II - Analyse de code (···/3,5pt)

Nous considérons une sous routine permettant de calculer une fonction mathématique. Le code obtenu après compilation avec optimisations est présenté ci-dessous. Les paramètres sont passés par $r0$ et $r1$ et le résultat est renvoyé dans $r0$.

```
mul     r3, r0, r0
add     r1, r1, r1, lsl #1
add     r3, r1, r3, lsl #1
add     r3, r3, r0
mov     r0,r3
```

$r3 \leftarrow r0^2$
$r1 \leftarrow 3r1$
$r3 \leftarrow 2r0^2 + 3r1$
$r3 \in 2r0^2 + 3r1 + r0$.
$r0 \in 2r0^2 + 3r1 + r0$

Q.2.4) Après avoir analysé cette suite d'instructions, déterminer la fonction mathématique réalisée.

Le code obtenu sans optimisation est présenté en annexe. La pile (SP) pointe à l'adresse 0x1000 avant l'appel de cette routine.

Q.2.5) Compléter la partie (a) de l'annexe pour connaître le contenu de la pile après l'exécution de chaque instruction.

Q.2.6) Compléter la partie (b) pour représenter l'évolution du contenu de la pile.

Instructions		SP	FP
mov	ip, sp	0x1000	
stmfd	sp!, {fp, ip, lr, pc}		
sub	fp, ip, #4		0x0FFC
sub	sp, sp, #12	0x0FF4	
str	r0, [fp, #-16]		
str	r1, [fp, #-20]		
ldr	r2, [fp, #-16]		
ldr	r3, [fp, #-16]		
mul	r3, r2, r3		
mov	r1, r3, lsl #1		
ldr	r2, [fp, #-20]		
mov	r3, r2		
mov	r3, r3, lsl #1		
add	r3, r3, r2		
add	r2, r1, r3		
ldr	r3, [fp, #-16]		
add	r3, r2, r3		
str	r3, [fp, #-24]		
ldr	r3, [fp, #-24]		
mov	r0, r3		
ldmea	fp, {fp, sp, pc}		

(a) code obtenu sans optimisation

Adresse	
0x0FE0	
0x0FE4	r_3
0x0FE8	r_1
0x0FEC	r_0
0x0FF0	fp
0x0FF4	IP = 0x1000
0x0FF8	LR
fp → 0x0FFC	PC
IP → 0x1000	
0x1004	
0x1008	
0x100C	
0x1010	
0x1014	
0x1018	
0x101c	

(b) contenu de la pile